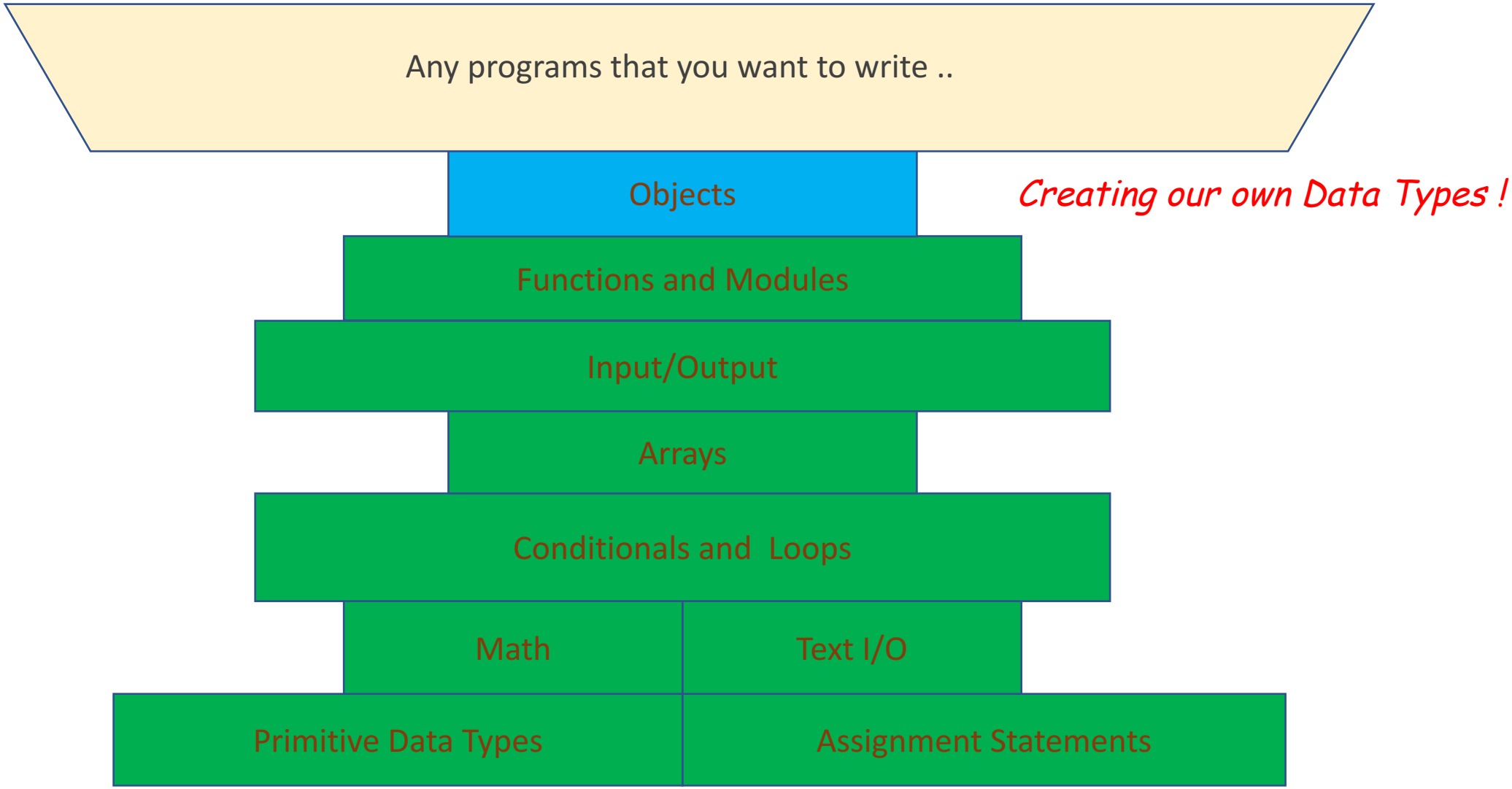
Reviews: ADTs, Classes, Interfaces

CSC 209 Data Structures

Kulwadee Somboonviwat

kulwadee.som [at] sit.kmutt.ac.th

Basic Building Blocks for Programming (in High-level language)



Lecture Plan

- Primitive Types vs. Reference Types
- Abstract Data Types (ADT)
- Object-oriented programming (OOP)
- Using data types
- Implementing data types
- Interfaces

Primitive Types vs Reference Types

- **Primitive Types**
  - boolean (1B), byte (1B), char (2B), short (2B), int (4B), long (8B), float (4B), double (8B)
  - Variable holds one value of given type
  - boolean instance variables, initialized to false
  - Other primitive instance variables initialized to zero
  - Do not have methods

Primitive Types vs Reference Types

- **Reference Types**
  - Any non-primitive type is a reference type
  - Variables refer to objects that may contain many instance variables
  - Default instance variable value is null – reference to nothing
  - Reference required to call an object’s methods

Abstract Data Types

- Data Type. Set of values and operations on those values.
- Primitive types (aka, Built-in types)
  - Values immediately map to machine representations
  - Operations immediately map to machine instructions

Data Type	Set of Values	Operations
`boolean`	`true, false`	not, and, or, xor
`int`	-2^{31} to $2^{31} - 1$	add, subtract, multiply
`double`	any of 2^{64} possible reals	add, subtract, multiply

- We want to write programs that process other types of data.
  - Colors, String, complex numbers, vectors, matrices, Points, ...
- An abstract data type is a data type whose representation is hidden from the client.

Object-oriented programming (OOP)

- Object-oriented programming (OOP).
  - Create your own data types.
  - Use them in your programs (manipulate objects).
    - Create an **object** that holds a data type value.
    - Refer to the object via a **variable**.
- Examples:

Data type	Set of values	Examples of operations
Color	Three 8-bit integers	Get red component, brighten, darken
Picture	2D array of colors	Get/set color of pixels (i,j)
String	Sequence of characters	lengths, substring, compare

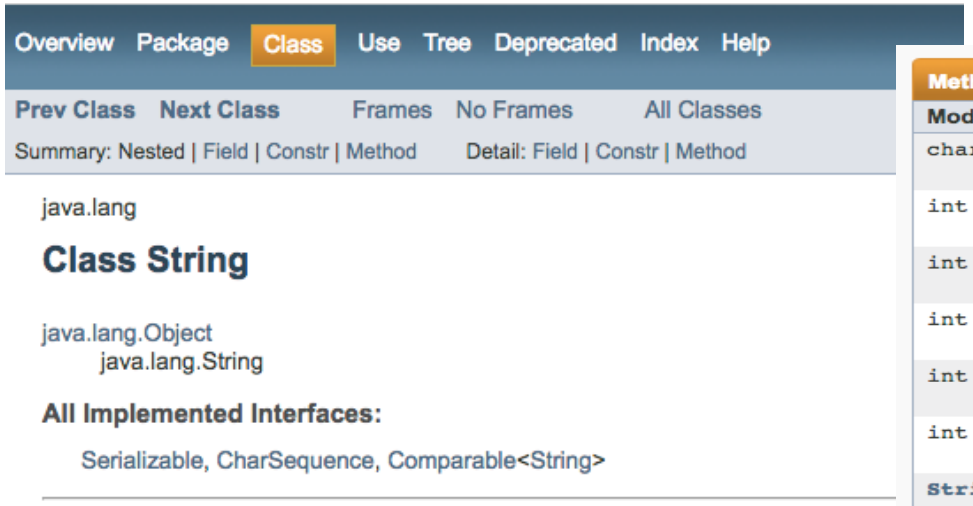
- Clients can use ADTs without knowing implementation details.
  - Just like clients can use functions without knowing implementation details.

Strings

- A String is a sequence of characters.

values

- Java's String ADT allows us to write Java program that manipulate strings.
  - The exact representation is hidden from us (it could change and our program would still work)



(operations)

Methods

Modifier and Type	Method and Description
char	**charAt**(int index) Returns the char value at the specified index.
int	**codePointAt**(int index) Returns the character (Unicode code point) at the specified index.
int	**codePointBefore**(int index) Returns the character (Unicode code point) before the specified index.
int	**codePointCount**(int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this String.
int	**compareTo**(String anotherString) Compares two strings lexicographically.
int	**compareToIgnoreCase**(String str) Compares two strings lexicographically, ignoring case differences.
String	**concat**(String str) Concatenates the specified string to the end of this string.
boolean	**contains**(CharSequence s) Returns true if and only if this string contains the specified sequence of char values.
boolean	**contentEquals**(CharSequence cs) Compares this string to the specified CharSequence.
boolean	**contentEquals**(StringBuffer sb) Compares this string to the specified StringBuffer.
static String	**copyValueOf**(char[] data) Returns a String that represents the character sequence in the array specified.
static String	**copyValueOf**(char[] data, int offset, int count) Returns a String that represents the character sequence in the array specified.
boolean	**endsWith**(String suffix) Tests if this string ends with the specified suffix.
boolean	**equals**(Object anObject) Compares this string to the specified object.

https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

Using ADTs

- To use an ADT, you need to know:
  - Its name
  - How to *construct* new objects.
  - How to *apply operations* to a given object.

- To construct a new object
  - Use the keyword *new* to invoke a *constructor.*
  - Use the *data type name* to specify type of object.

- To apply an operation (invoke a method)
  - Use *object name* to specify which object.
  - Use the *dot operator* to indicate that an operation is to be applied.
  - Use a *method name* to specify which operation.

```
String s = new String (”Hello World”);
s.substring(0, 5);
```

Using String ADT

public class String *(Java string data type)*	
String(String s)	*create a string with the same value as* **s**
String(char[] a)	*create a string that represents the same sequence of characters as in* **a[]**
int length()	*number of characters*
char charAt(int i)	*the character at index* **i**
String substring(int i, int j)	*characters at indices* **i** *through* **(j-1)**
boolean contains(String substring)	*does this string contain* **substring** ?
boolean startsWith(String pre)	*does this string start with* **pre** ?
boolean endsWith(String post)	*does this string end with* **post** ?
int indexOf(String pattern)	*index of first occurrence of* **pattern**
int indexOf(String pattern, int i)	*index of first occurrence of* **pattern** *after* **i**
String concat(String t)	*this string with* **t** *appended*
int compareTo(String t)	*string comparison*
String toLowerCase()	*this string, with lowercase letters*
String toUpperCase()	*this string, with uppercase letters*
String replaceAll(String a, String b)	*this string, with* **a***s replaced by* **b***s*
String[] split(String delimiter)	*strings between occurrences of* **delimiter**
boolean equals(Object t)	*is this string's value the same as* **t***'s* ?
int hashCode()	*an integer hash code*

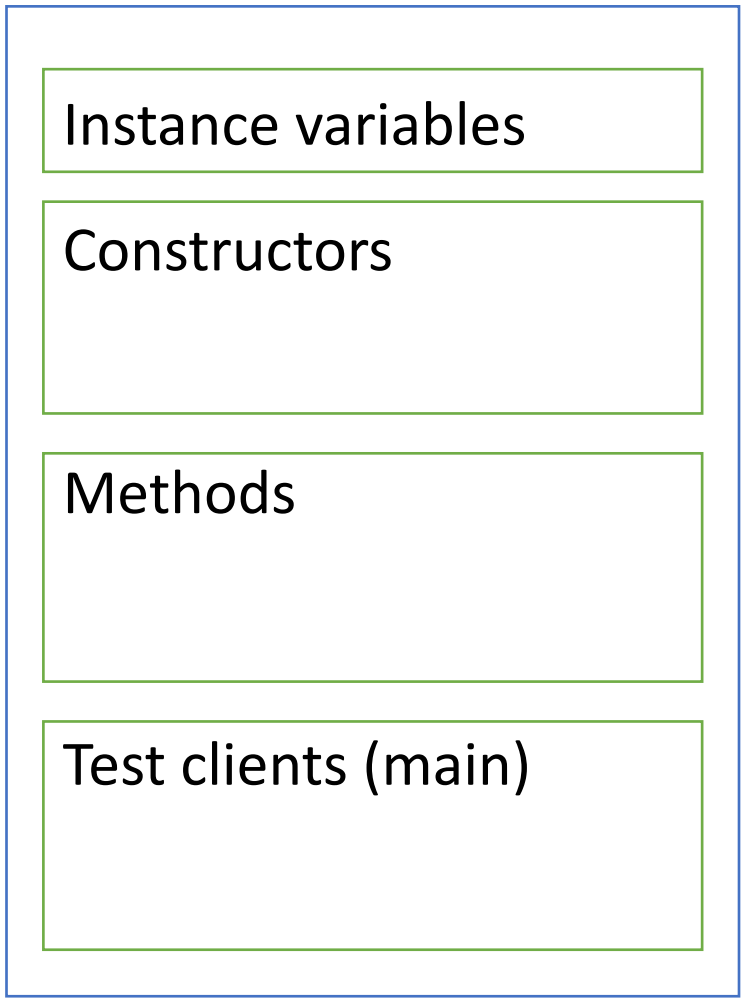
```java
/**
 * Returns true if s is a Palindrome (s reads the same forward/backward.
 */
public static boolean isPalindrome(String s) {
    s = s.toLowerCase();
    int N = s.length();
    for (int i = 0; i < N/2; i++)
        if (s.charAt(i) != s.charAt(N-1-i))
                return false;
    return true;
}

/**
 * Find lines containing a specified string (query) in Standard input
 */
public static void findMatches(String query) {
    Scanner in = new Scanner(System.in);
    while (in.hasNextLine()) {
        String s = in.nextLine();
        if (s.contains(query))
            System.out.println(s);
    }
}
```

Implementing an ADT

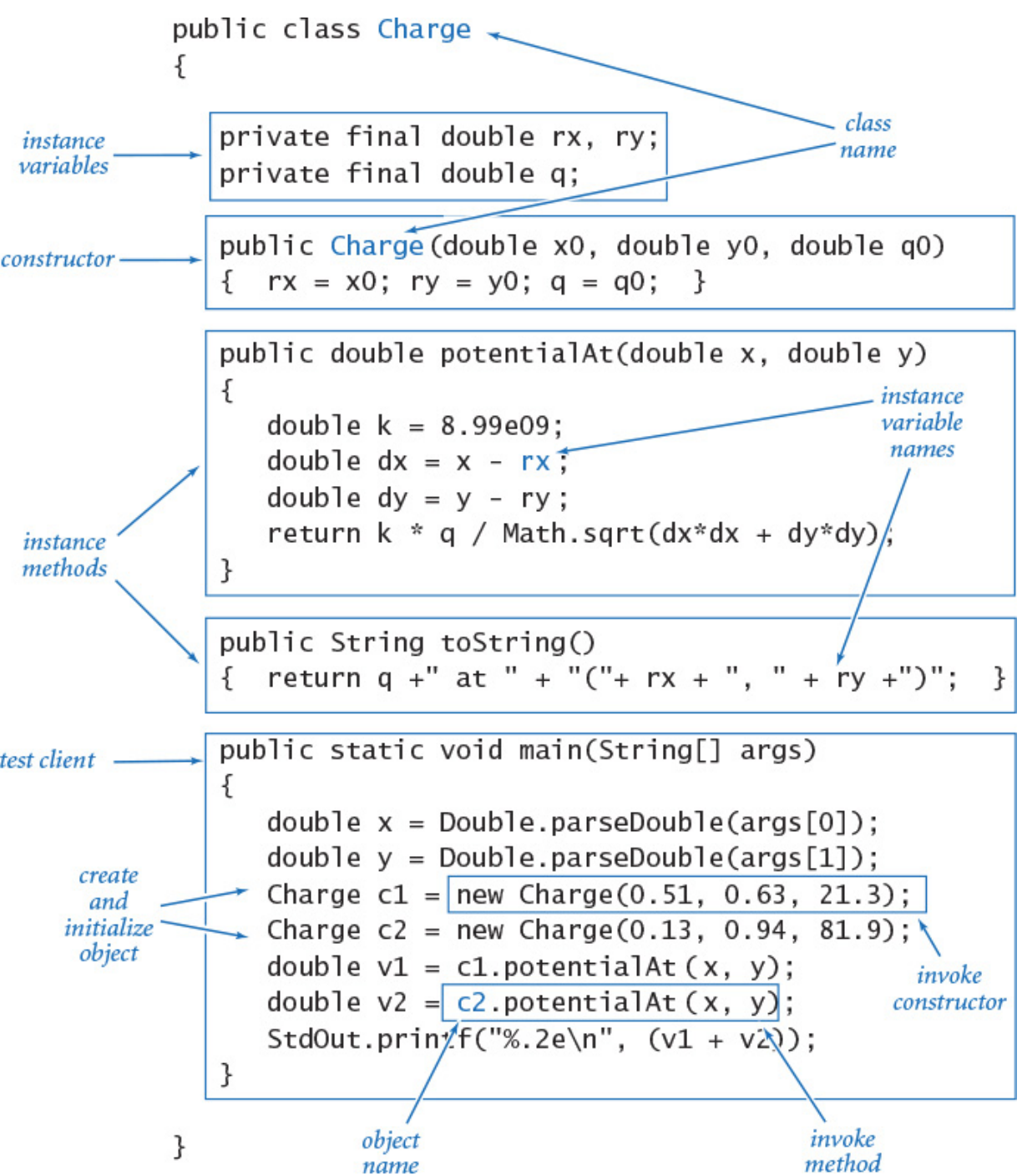
In Java, an implementation of a data type is called a class

- To create a data type. You need to provide code that
  - Defines the set of values (instance variables).
  - Implements operations on those values (methods).
  - Create and initialize new objects (constructors).

- Instance variables
  - Declarations associate variable names with types
  - Set of type values is “set of values”.
- Methods
  - Like static methods.
  - Can refer to instance variables.
- Constructors
  - Like a method with the same name as the type.
  - No return type declaration.
  - Invoke by **new**, return object of the type.



An anatomy of a Java Class



Complex Number Data Type

- Goal. Create a data type to manipulate complex numbers.
- Set of values. Two real numbers: real and imaginary parts.
- Operations. (API)

public class Complex		
	Complex(double real, double imag)	
Complex	plus(Complex b)	*sum of this number and* b
Complex	times(Complex b)	*product of this number and* b
double	abs()	*magnitude*
String	toString()	*string representation*

$a = 3 + 4i,\ b = -2 + 3i$

$a + b = 1 + 7i$

$a \times b = -18 + i$

$|a| = 5$

Complex Number Data Type: a Client program

```
public static void main(String[] args) {
   Complex a = new Complex( 3.0, 4.0);
   Complex b = new Complex(-2.0, 3.0);
   Complex c = a.times(b);
   StdOut.println("a = " + a);
   StdOut.println("b = " + b);
   StdOut.println("c = " + c);
}
```

result of `c.toString()`

```
% java TestClient
a = 3.0 + 4.0i
b = -2.0 + 3.0i
c = -18.0 + 1.0i
```

Complex Number Class (an implementation of ADT)

```
public class Complex {

   private final double re;
   private final double im;                          instance variables

   public Complex(double real, double imag) {
      re = real;
      im = imag;
   }                                                 constructor

   public String toString() { return re + " + " + im + "i"; }

   public double abs() { return Math.sqrt(re*re + im*im); }

   public Complex plus(Complex b) {
      double real = re + b.re;
      double imag = im + b.im;                       creates a Complex object,
      return new Complex(real, imag);                and returns a reference to it
   }

   public Complex times(Complex b) {                 refers to b's instance variable
      double real = re * b.re - im * b.im;
      double imag = re * b.im + im * b.re;
      return new Complex(real, imag);
   }                                                 methods
}
```

Vector Data Type

- Set of values. Sequence of real numbers. [Cartesian coordinates]
- Operations. (API)

public class Vector		
	`Vector(double[] a)`	*create a vector with the given Cartesian coordinates*
`Vector`	`plus(Vector b)`	*sum of this vector and* `b`
`Vector`	`minus(Vector b)`	*difference of this vector and* `b`
`Vector`	`times(double t)`	*scalar product of this vector and* `t`
`double`	`dot(Vector b)`	*dot product of this vector and* `b`
`double`	`magnitude()`	*magnitude of this vector*
`Vector`	`direction()`	*unit vector with same direction as this vector*

$$x = (0, 3, 4, 0), \quad y = (0, -3, 1, -4)$$

$$x + y = (0, 0, 5, -4)$$

$$3x = (0, 9, 12, 0)$$

$$x \cdot y = (0 \cdot 0) + (3 \cdot -3) + (4 \cdot 1) + (0 \cdot -4) = -5$$

$$|x| = (0^2 + 3^2 + 4^2 + 0^2)^{1/2} = 5$$

$$\vec{x} = x / |x| = (0, 0.6, 0.8, 0)$$

Practice Exercise

- Complete the implementation of the **Vector** class.

  (the initial code is provided in the course GitHub)

Interfaces

- A set of requirements for classes.
- Class can choose to conform to one or more interfaces.
- Usages:
  Service provider: “if your class conforms to a particular interface, then I’ll perform the service.”
  - E.g. Arrays.sort sorts an array if the element class conforms to the Comparable interface
- Example definition:

```
public interface Comparable<T> {
    /**...*/
    public int compareTo(T o);
}
```

- Conforming class must provide **compareTo** method

Interfaces

- A class can choose to *implement* one or more interfaces:

```
public class AccountInf implements Comparable<AccountInf> {

    @Override
    /**
     * Compares accounts by amount value.
     * @param other another AccountInf object
     * @return a positive integer if other should come before this object.
     *         Zero if the two are indistinguishable (equal).
     *         a negative integer otherwise.
     */
    public int compareTo(AccountInf other) {
        return Double.compare(amount, other.amount);
    }

}
```

Interfaces

- Using the interface: a sorting algorithm keeps comparing elements and rearranges them if they are out of order:

```
public static void sort(Comparable[] a) {
      // ….
      if ( a[i].compareTo( a[j] ) > 0) {
            // swap accounts[i] and accounts[j]
      }
      // …
}
```

Interfaces: a client program

```
package co.kulwadee.csc209.lect01;

import java.util.Arrays;

public class AccountInfClient {
    public static void main(String[] args) {
        AccountInf[] accounts = new AccountInf[3];

        accounts[0] = new AccountInf("A", 200);
        accounts[1] = new AccountInf("B", 100);
        accounts[2] = new AccountInf("C", 300);

        System.out.println("Before sorting...");

        for (AccountInf acc : accounts)
            System.out.println(acc);

        Arrays.sort(accounts);

        System.out.println("After sorting...");
        for (AccountInf acc : accounts)
            System.out.println(acc);
    }

}
```

Summary

- Object-oriented programming.
  - Create our own data types (with representation hidden from clients – ADT)
  - Use them in our programs (manipulate objects via *reference*).

- In Java, programs manipulate objects via references
  - String, Arrays, user-defined ADTs are *reference types*
  - Exceptions: boolean, int, double, char, and other are primitive types

- An implementation of ADT is called a class. To define a class, you need
  - Instance variables
  - Methods
  - Constructors

Summary

- **Interfaces**
  - An **interface** allows us to specify what an object of a given type must do.
    - Conforming class must implement all methods specify by the interface.
  - A class can choose to implement any number of interfaces.
  - Used extensively in the Java Collection Framework (JCF)